Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2343 – Arquitectura de Computadores

Ayudantía 2

Profesor: Yadran Eterovic

Ayudantes: Daniel Leal (dlleal@uc.cl), Jessica Hormazabal (jyhormazabal@uc.cl)

Preguntas

Los ejercicios de esta ayudantía fueron resueltos utilizando la arquitectura ISA MEEP MEEP (solamente 4 registros s1, s2, s3, s4)y no la arquitectura ISA MIPS. Hay instrucciones que difieren de ambas arquitecturas, por lo tanto es importante que se fijen en la ISA que van a usar antes de escribir un programa. Notar además que las soluciones para esta ayudantía son soluciones posibles pero están lejos de ser las únicas soluciones válidas.

1. **Assembly**

   a) Cargue los valores 5 y 17 en los registros $s1 y $s2. Considere registros de 1 byte.

```
movi $s1, 5   # s1 = 5
movi $s2, 17  # s2 = 17
```

   TIP: Para resolver este ejercicio con la ISA MIPS, se podría hacer lo siguiente:

```
addi $s1, $zero, 5   # s1 = 5
addi $s2, $zero, 17  # s2 = 17
```

   b) Utilizando la arquitectura ISA MEEP MEEP, programe una multiplicación en *assembly* entre dos números guardados en memoria. Considere que los números a multiplicar están en las direcciones 8 y 9, mientras que el resultado desea guardarse en la dirección 13.

```
movi $s2, 8          # s2 = 8
movi $s3, 9          # s3 = 9
lb   $s1, 0($s2)     # s1 = Memory[8]
lb   $s2, 0($s3)     # s2 = Memory[9]
movi $s3, 0          # s3 = 0 (contador)
movi $s4, 0          # s4 = 0 (acumulado)
beq  $s2, $s3, 4     # si contador == s2, saltar a 11 (7 + 4)
addi $s3, $s3, 1     # s3 += 1 (aumentar contador)
add  $s4, $s4, $s1   # s4 += s1
j    7               # saltar a 7
movi $s1, 13         # s1 = 13 (queremos escribir en Memory[13])
sb   $s3, 0($s1)     # Memory[13] = s3
```

c) Utilizando la arquitectura ISA MEEP MEEP, programe en *assembly* un código que le permita sumar N números pares positivos desde el número P_0 en adelante. Asuma que N y P_0 están guardados en la memoria RAM en las direcciones 13 y 21, respectivamente. Guarde el valor final en la dirección 4 de la memoria. Comente si lo encuentra necesario.

Lo primero que haremos será definir para qué usaremos cada registro. Usaremos s1 como un contador para saber qué número estamos sumando, s2 lo usaremos para saber hasta qué número sumar, s3 lo usaremos como un extractor temporal (un recipiente en el cual almacenar temporalmente los números extraídos de memoria) y s4 lo usaremos para almacenar el total sumado. Una vez definido esto, podemos empezar a escribir el programa.

```
1  movi $s2, 21          # s2 = 21
2  lb   $s1, 0($s2)      # s1 = Memory[21] (ubicacion del primer numero a sumar)
3  movi $s2, 13          # s2 = 13
4  lb   $s3, 0($s2)      # s3 = Memory[13] (cantidad de numeros a sumar)
5  add  $s2, $s1, $s3    # s2 = s1 + s3 (ubicacion del ultimo numero a sumar)
6  movi $s4, 0           # s4 = 0 (valor total sumado)
7  beq  $s1, $s2, 5      # si s1 == s2, saltar a 12 (7 + 5)
8  lb   $s3, 0($s1)      # s3 = Memory[s1]
9  add  $s4, $s4, $s3    # s4 += s3
10 addi $s1, $s1, 1      # s1 += 1 (contador)
11 j    7                # saltar a 7
12 movi $s1, 4           # s1 = 4 (queremos escribir en Memory[4])
13 sb   $s4, 0($s1)      # Memory[4] = s4
```

2. **Anexos**

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

Figura 1: ISA de un computador MIPS

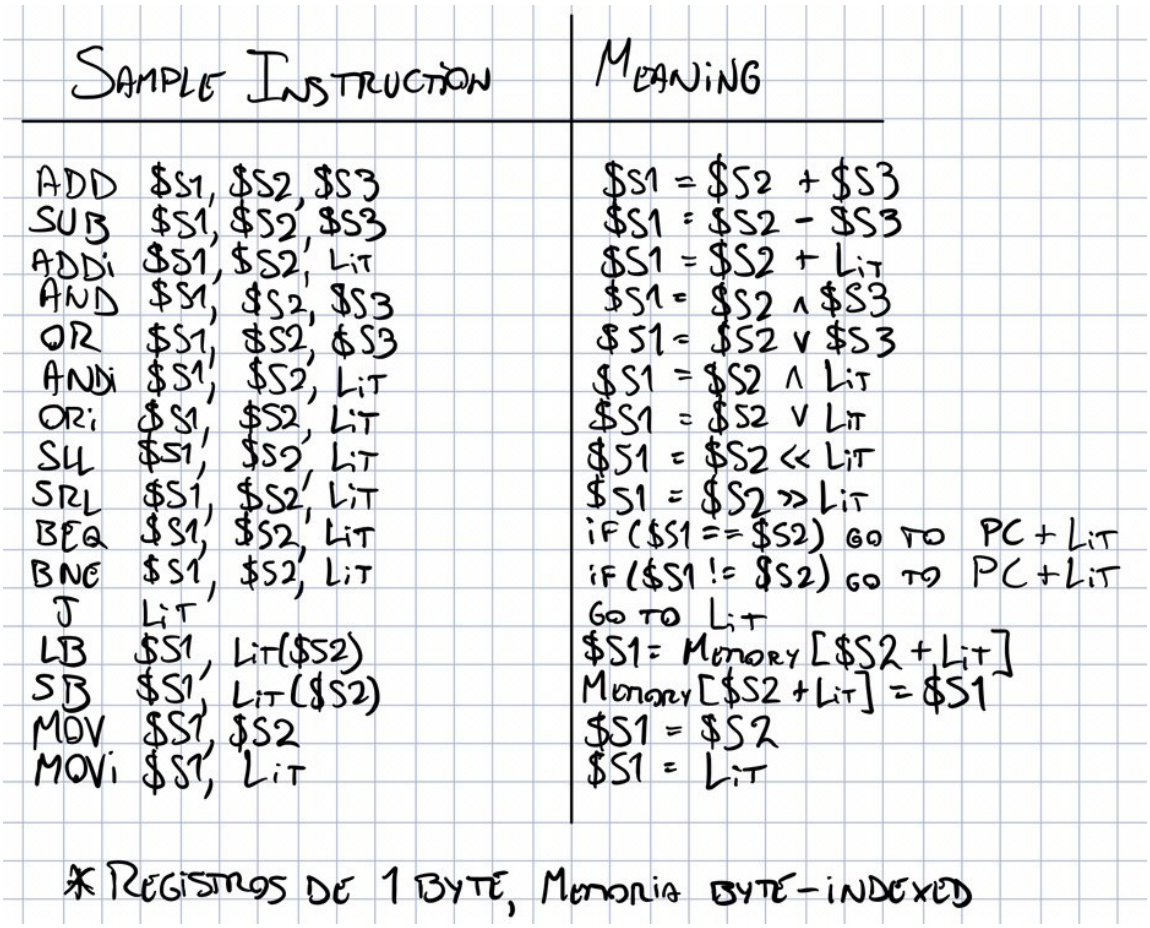

Figura 2: ISA de un computador MEEP MEEP